In [4]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\15513\appdata\local\python\pythonc
ore-3.14-64\lib\site-packages (2.3.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\15513\appdata\local\python
\pythoncore-3.14-64\lib\site-packages (from pandas) (2.3.5)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\15513\appdata\loca
l\python\pythoncore-3.14-64\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\15513\appdata\local\python\p
ythoncore-3.14-64\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\15513\appdata\local\python
\pythoncore-3.14-64\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\15513\appdata\local\python\pytho
ncore-3.14-64\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

In [8]:
```
pip install mysql-connector-python
```

```
Collecting mysql-connector-python
  Using cached mysql_connector_python-9.5.0-cp314-cp314-win_amd64.whl.metadata (7.7
kB)
Using cached mysql_connector_python-9.5.0-cp314-cp314-win_amd64.whl (17.0 MB)
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.5.0
Note: you may need to restart the kernel to use updated packages.
```

In [12]:
```
pip install seaborn
```

```
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\15513\appdata\local
\python\pythoncore-3.14-64\lib\site-packages (from seaborn) (2.3.5)
Requirement already satisfied: pandas>=1.2 in c:\users\15513\appdata\local\python\py
thoncore-3.14-64\lib\site-packages (from seaborn) (2.3.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\15513\appdata\loc
al\python\pythoncore-3.14-64\lib\site-packages (from seaborn) (3.10.7)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\15513\appdata\local\pyth
on\pythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.
3.3)
Requirement already satisfied: cycler>=0.10 in c:\users\15513\appdata\local\python\p
ythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\15513\appdata\local\pyt
hon\pythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.
61.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\15513\appdata\local\pyt
hon\pythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.
4.9)
Requirement already satisfied: packaging>=20.0 in c:\users\15513\appdata\local\pytho
n\pythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (25.
0)
Requirement already satisfied: pillow>=8 in c:\users\15513\appdata\local\python\pyth
oncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (12.0.0)
Requirement already satisfied: pyparsing>=3 in c:\users\15513\appdata\local\python\p
ythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\15513\appdata\local
\python\pythoncore-3.14-64\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\15513\appdata\local\python\p
ythoncore-3.14-64\lib\site-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\15513\appdata\local\python
\pythoncore-3.14-64\lib\site-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\15513\appdata\local\python\pytho
ncore-3.14-64\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4-
>seaborn) (1.17.0)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```python
import pandas as pd
import mysql.connector
import os

# List of CSV files and their corresponding table names
csv_files = [
    ('customers.csv', 'customers'),
    ('orders.csv', 'orders'),
    ('sellers.csv', 'sellers'),
    ('products.csv', 'products'),
    ('geolocation.csv', 'geolocation'),
    ('payments.csv', 'payments'),
    ('order_items.csv','order_items')# Added payments.csv for specific handling
]
```

```
# Connect to the MySQL database
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='Saiprakash@9798',
    database='ecommerce'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'C:/Users/15513/Desktop/project2'

def get_sql_type(dtype):
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'FLOAT'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATETIME'
    else:
        return 'TEXT'

for csv_file, table_name in csv_files:
    file_path = os.path.join(folder_path, csv_file)

    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Replace NaN with None to handle SQL NULL
    df = df.where(pd.notnull(df), None)

    # Debugging: Check for NaN values
    print(f"Processing {csv_file}")
    print(f"NaN values before replacement:\n{df.isnull().sum()}\n")

    # Clean column names
    df.columns = [col.replace(' ', '_').replace('-', '_').replace('.', '_') for col

    # Generate the CREATE TABLE statement with appropriate data types
    columns = ', '.join([f'`{col}` {get_sql_type(df[col].dtype)}' for col in df.col
    create_table_query = f'CREATE TABLE IF NOT EXISTS `{table_name}` ({columns})'
    cursor.execute(create_table_query)

    # Insert DataFrame data into the MySQL table
    for _, row in df.iterrows():
        # Convert row to tuple and handle NaN/None explicitly
        values = tuple(None if pd.isna(x) else x for x in row)
        sql = f"INSERT INTO `{table_name}` ({', '.join(['`' + col + '`' for col in
        cursor.execute(sql, values)

    # Commit the transaction for the current CSV file
    conn.commit()
```

```
# Close the connection
conn.close()
```

```
Processing customers.csv
NaN values before replacement:
customer_id                 0
customer_unique_id          0
customer_zip_code_prefix    0
customer_city               0
customer_state              0
dtype: int64

Processing orders.csv
NaN values before replacement:
order_id                            0
customer_id                         0
order_status                        0
order_purchase_timestamp            0
order_approved_at                 160
order_delivered_carrier_date     1783
order_delivered_customer_date    2965
order_estimated_delivery_date       0
dtype: int64

Processing sellers.csv
NaN values before replacement:
seller_id                 0
seller_zip_code_prefix    0
seller_city               0
seller_state              0
dtype: int64

Processing products.csv
NaN values before replacement:
product_id                      0
product category              610
product_name_length           610
product_description_length    610
product_photos_qty            610
product_weight_g                2
product_length_cm               2
product_height_cm               2
product_width_cm                2
dtype: int64

Processing geolocation.csv
NaN values before replacement:
geolocation_zip_code_prefix    0
geolocation_lat                0
geolocation_lng                0
geolocation_city               0
geolocation_state              0
dtype: int64

Processing payments.csv
NaN values before replacement:
order_id                0
payment_sequential      0
payment_type            0
```

```
payment_installments    0
payment_value           0
dtype: int64

Processing order_items.csv
NaN values before replacement:
order_id                0
order_item_id           0
product_id              0
seller_id               0
shipping_limit_date     0
price                   0
freight_value           0
dtype: int64
```

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector


db = mysql.connector.connect(host = "localhost",
                             username = "root",
                             password = "Saiprakash@9798",
                             database = "ecommerce")

cur = db.cursor()
```

List all unique cities where customers are located.

In [4]:
```
query = """ select distinct customer_city from customers """

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data)
df.head()
```

Out[4]:

	0
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruzes
4	campinas

Count the number of orders placed in 2017.

In [5]:
```
query = """ select count(order_id) from orders where year(order_purchase_timestamp)

cur.execute(query)

data = cur.fetchall()

"total orders placed in 2017 are", data[0][0]
```

Out[5]:
```
('total orders placed in 2017 are', 135303)
```

Find the total sales per category.

In [6]:
```
query = """ select upper(products.product_category) category,
round(sum(payments.payment_value),2) sales
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category
"""

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data, columns = ["Category", "Sales"])
df
```

Out[6]:

	Category	Sales
0	PERFUMERY	13681943.82
1	FURNITURE DECORATION	38614762.58
2	TELEPHONY	13145815.37
3	BED TABLE BATH	46238949.12
4	AUTOMOTIVE	23011946.94
...	...	...
69	CDS MUSIC DVDS	32384.61
70	LA CUISINE	78665.31
71	FASHION CHILDREN'S CLOTHING	21213.09
72	PC GAMER	58709.61
73	INSURANCE AND SERVICES	8761.77

74 rows × 2 columns

Calculate the percentage of orders that were paid in installments.

In [7]:
```
query = """ select ((sum(case when payment_installments >= 1 then 1
else 0 end))/count(*))*100 from payments
"""

cur.execute(query)

data = cur.fetchall()

"the percentage of orders that were paid in installments is", data[0][0]
```

Out[7]:
```
('the percentage of orders that were paid in installments is',
 Decimal('99.9981'))
```

Count the number of customers from each state.

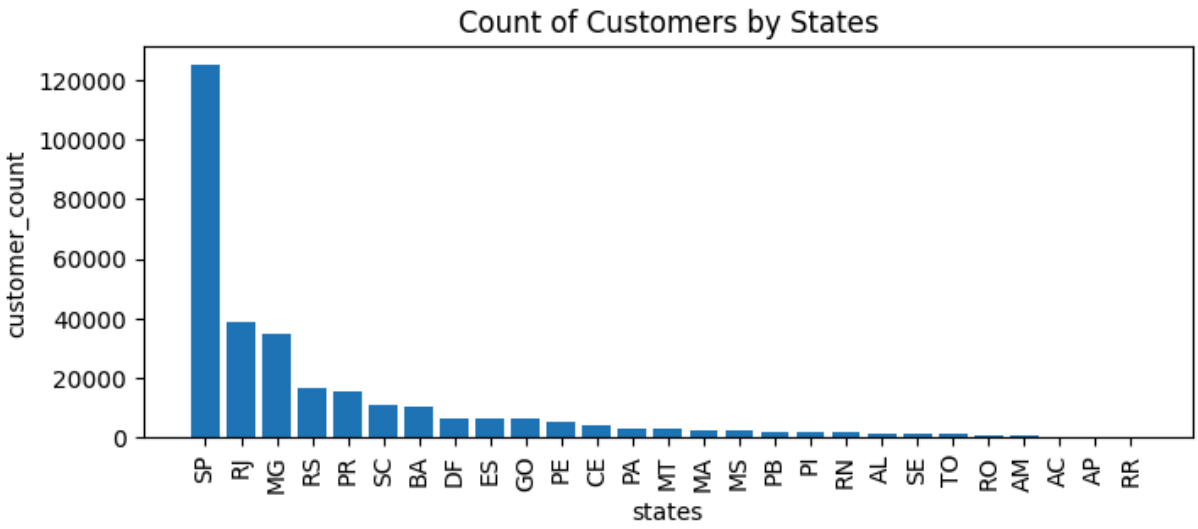
In [8]:
```
query = """ select customer_state ,count(customer_id)
from customers group by customer_state
"""

cur.execute(query)

data = cur.fetchall()
```

```
df = pd.DataFrame(data, columns = ["state", "customer_count" ])
df = df.sort_values(by = "customer_count", ascending= False)

plt.figure(figsize = (8,3))
plt.bar(df["state"], df["customer_count"])
plt.xticks(rotation = 90)
plt.xlabel("states")
plt.ylabel("customer_count")
plt.title("Count of Customers by States")
plt.show()
```



Calculate the number of orders per month in 2018.

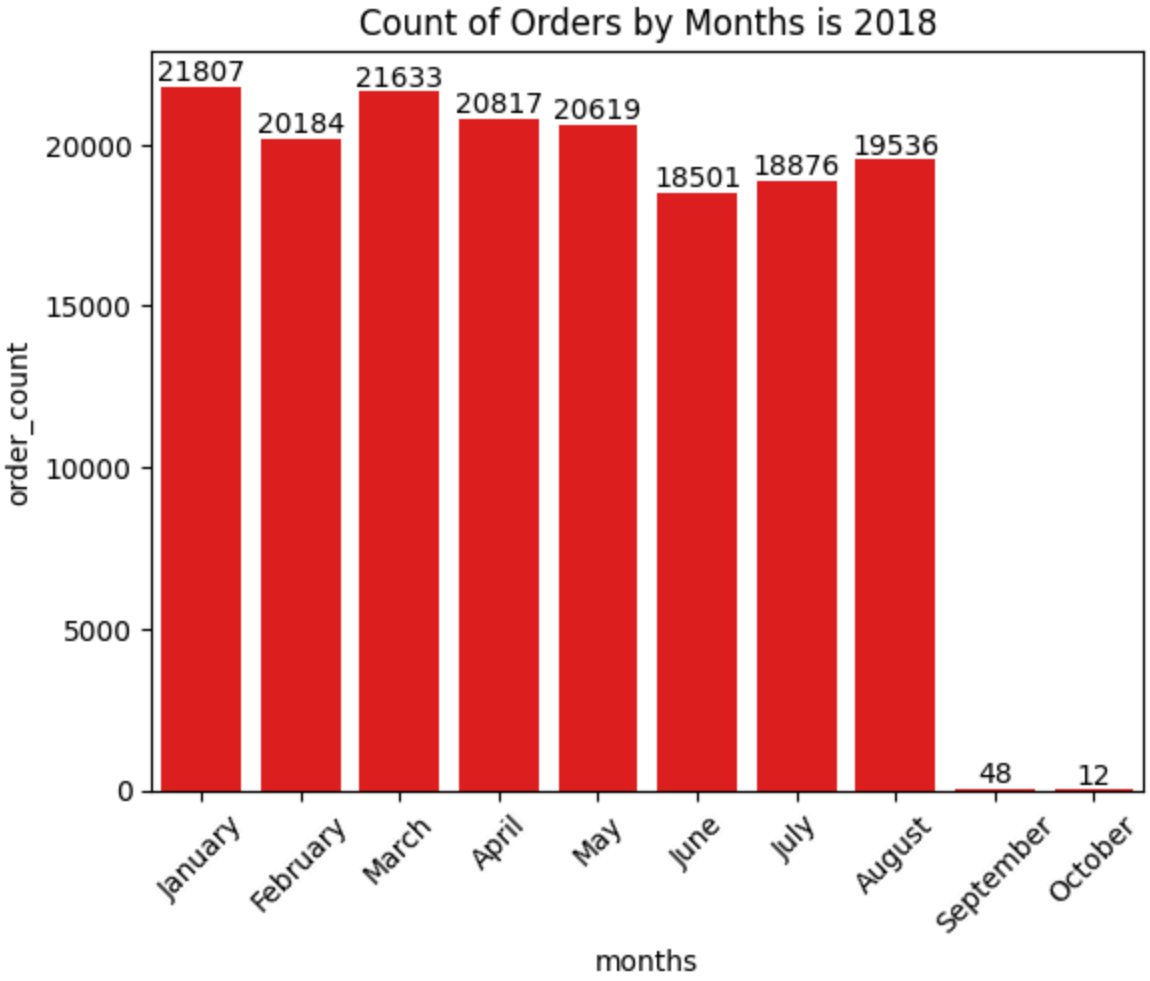
In [9]:
```
query = """ select monthname(order_purchase_timestamp) months, count(order_id) orde
from orders where year(order_purchase_timestamp) = 2018
group by months
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["months", "order_count"])
o = ["January", "February","March","April","May","June","July","August","September"

ax = sns.barplot(x = df["months"],y =  df["order_count"], data = df, order = o, col
plt.xticks(rotation = 45)
ax.bar_label(ax.containers[0])
plt.title("Count of Orders by Months is 2018")

plt.show()
```



Find the average number of products per order, grouped by customer city.

In [10]:
```
query = """with count_per_order as
(select orders.order_id, orders.customer_id, count(order_items.order_id) as oc
from orders join order_items
on orders.order_id = order_items.order_id
group by orders.order_id, orders.customer_id)

select customers.customer_city, round(avg(count_per_order.oc),2) average_orders
from customers join count_per_order
on customers.customer_id = count_per_order.customer_id
group by customers.customer_city order by average_orders desc
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data,columns = ["customer city", "average products/order"])
df.head(10)
```

Out[10]:

	customer city	average products/order
0	padre carvalho	63.00
1	celso ramos	58.50
2	datas	54.00
3	candido godoi	54.00
4	matias olimpio	45.00
5	cidelandia	36.00
6	curralinho	36.00
7	picarra	36.00
8	morro de sao paulo	36.00
9	teixeira soares	36.00

Calculate the percentage of total revenue contributed by each product category.

In [11]:

```
query = """select upper(products.product_category) category,
round((sum(payments.payment_value)/(select sum(payment_value) from payments))*100,2
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category order by sales_percentage desc"""


cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns = ["Category", "percentage distribution"])
df.head()
```

Out[11]:

	Category	percentage distribution
0	BED TABLE BATH	96.28
1	HEALTH BEAUTY	93.18
2	COMPUTER ACCESSORIES	89.13
3	FURNITURE DECORATION	80.40
4	WATCHES PRESENT	80.35

Calculate the total revenue generated by each seller, and rank them by revenue.

In [ ]:
```
query = """ select *, dense_rank() over(order by revenue desc) as rn from
(select order_items.seller_id, sum(payments.payment_value)
revenue from order_items join payments
on order_items.order_id = payments.order_id
group by order_items.seller_id) as a """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["seller_id", "revenue", "rank"])
df = df.head()
sns.barplot(x = "seller_id", y = "revenue", data = df)
plt.xticks(rotation = 90)
plt.show()
```

Calculate the moving average of order values for each customer over their order history.

In [ ]:
```
query = """select customer_id, order_purchase_timestamp, payment,
avg(payment) over(partition by customer_id order by order_purchase_timestamp
rows between 2 preceding and current row) as mov_avg
from
(select orders.customer_id, orders.order_purchase_timestamp,
payments.payment_value as payment
from payments join orders
on payments.order_id = orders.order_id) as a"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data)
df
```

Calculate the cumulative sales per month for each year.

In [ ]:
```
query = """select years, months , payment, sum(payment)
over(order by years, months) cumulative_sales from
(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years, months order by years, months) as a
"""
cur.execute(query)
```

```
data = cur.fetchall()
df = pd.DataFrame(data)
df
```

Calculate the year-over-year growth rate of total sales.

In [ ]:
```
query = """with a as(select year(orders.order_purchase_timestamp) as years,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years order by years)

select years, ((payment - lag(payment, 1) over(order by years))/
lag(payment, 1) over(order by years)) * 100 from a"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["years", "yoy % growth"])
df
```

Calculate the retention rate of customers, defined as the percentage of customers who make another purchase within 6 months of their first purchase.

In [ ]:
```
query = """with a as (select customers.customer_id,
min(orders.order_purchase_timestamp) first_order
from customers join orders
on customers.customer_id = orders.customer_id
group by customers.customer_id),

b as (select a.customer_id, count(distinct orders.order_purchase_timestamp) next_or
from a join orders
on orders.customer_id = a.customer_id
and orders.order_purchase_timestamp > first_order
and orders.order_purchase_timestamp <
date_add(first_order, interval 6 month)
group by a.customer_id)

select 100 * (count( distinct a.customer_id)/ count(distinct b.customer_id))
from a left join b
on a.customer_id = b.customer_id ;"""

cur.execute(query)
data = cur.fetchall()

data
```

Identify the top 3 customers who spent the most money in each year.

In [ ]:
```
query = """select years, customer_id, payment, d_rank
from
(select year(orders.order_purchase_timestamp) years,
orders.customer_id,
sum(payments.payment_value) payment,
dense_rank() over(partition by year(orders.order_purchase_timestamp)
order by sum(payments.payment_value) desc) d_rank
from orders join payments
on payments.order_id = orders.order_id
group by year(orders.order_purchase_timestamp),
orders.customer_id) as a
where d_rank <= 3 ;"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["years","id","payment","rank"])
sns.barplot(x = "id", y = "payment", data = df, hue = "years")
plt.xticks(rotation = 90)
plt.show()
```

In [ ]:
```

```

In [ ]:
```

```